TÉCNICO LISBOA

Aprendizagem 2021/22

Homework I – Group 72

I. Pen-and-paper

1)

The training results are as follows:

Class	prior probability
0	0.4
1	0.6

discrete attribute y2	x = 0	x = 1
P(y2 = A | Class = x)	2/4	1/6
P(y2 = B | Class = x)	1/4	1/3
P(y2 = C | Class = x)	1/4	1/2

Covariance indicates the level to which two variables vary together.

The covariance matrix of these two combined is as follows:

Covariance Matrix, Class = 0	{y3,y4} variable set	Covariance Matrix, Class = 1	{y3,y4} variable set
0.18	0.18	0.10966667	0.12233333
0.18	0.25	0.12233333	0.21366667

Numeric attributes

mean when class is zero(N):

$y1 = \frac{1}{4} * (0.6 + 0.1 + 0.2 + 0.1) = 0.24999999999999997$

$y3 = \frac{1}{4} * (0.2 - 0.1 - 0.1 + 0.8) = 0.2$

$y4 = \frac{1}{4} * (0.4 - 0.4 + 0.2 + 0.8) = 0.25$

mean when class is one(P):

$y1 = \frac{1}{6} * (0.3 - 0.1 - 0.3 + 0.2 + 0.4 - 0.2) = 0.049999999999999996$

$y3 = \frac{1}{6} * (0.1 + 0.2 + 0.1 + 0.5 - 0.4 + 0.4) = 0.11666666666666668$

$y4 = \frac{1}{6} * (0.3 - 0.2 + 0.2 + 0.6 - 0.7 + 0.3) = = 0.08333333333333333$

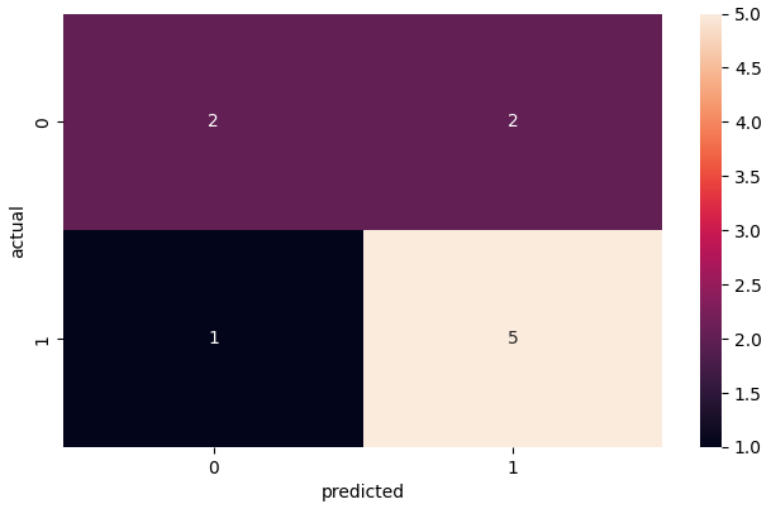
The variance of each individual variable sets was calculated as follows:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

Variable	*Class 1 (P)*	*Class 0 (N)*
y1	0.083	0.05666666666666665
y3	0.10966666666666666	0.18000000000000005
y4	0.21366666666666662	0.25000000000000006]

Homework I – Group 72

2)



In order to draw the confusion matrix, we had to compute several results.

1. Predictor Prior Probability using the normal pdf $N(x; \mu, \sigma)$. eg = P(feature | class). feature corresponds to the following variable sets {y1},{y2} and finally {y3,y4}.
2. Likelihood by getting the product of the prior and the Normal pdfs.
3. Joint Probability by Multiplying prior and the likelihood .

By calculating the joint probability to all y features in the 10 instances (rows) we obtained the following :

Probabilities	*P(y1|class = 0)*	*P(y2|class = 0)*	*P(y3,y4|class = 0)*	*P(y1|class = 1)*	*P(y2|class = 1)*	*P(y3,y4|class = 0)*
x1	0.5686237446173312	0.5	1.2073620797939855	0.2238519564702542	0.16666666666666666	1.2118617969040946
x2	1.374124251670586	0.25	0.46031334625225606	1.364050471665823	0.3333333333333333	0.9567056882218714
x3	1.6393293226109074	0.5	0.7066099904983237	1.2092213423146918	0.16666666666666666	0.6078648075567288
x4	1.374124251670586	0.25	0.5123716815646143	1.364050471665823	0.5	0.20301887559426582
x5	1.6393293226109071	0.25	1.1742857643081361	0.9502908135480692	0.3333333333333333	1.2071000671019052
x6	0.5686237446173312	0.25	0.33377892534405396	1.2092213423146918	0.5	0.6698137879639072
x7	0.1161617555326254	0.25	0.7066099904983237	0.6620375485160854	0.5	0.6078648075567288
x8	1.6393293226109074	0.25	1.0846908583841528	1.2092213423146918	0.3333333333333333	0.8408068361566832
x9	1.3741242516705858	0.5	0.21743662854985937	0.6620375485160852	0.16666666666666666	0.3880497696191463
x10	0.2807140695495067	0.25	1.0803950586304265	0.9502908135480692	0.5	1.1251825101728976

Homework I – Group 72

$P(X1,C=0)$	0.13730694938428503	$P(X1,C=1)$	0.027127763420853938
$P(X2,C=0)$	0.06325277324528648	$P(X2,C=1)$	0.2609989690528839
$P(X3,C=0)$	0.23167329541474335	$P(X3,C=1)$	0.07350430985396093
$P(X4,C=0)$	0.07040623535071755	$P(X4,C=1)$	0.083078397903427
$P(X5,C=0)$	0.19250410865548884	$P(X5,C=1)$	0.22941922096003967
$P(X6,C=0)$	0.018979462240348462	$P(X6,C=1)$	0.24298593833478124
$P(X7,C=0)$	0.008208105697317704	$P(X7,C=1)$	0.12072879810721762
$P(X8,C=0)$	0.1778165530117137	$P(X8,C=1)$	0.2033443142089507
$P(X9,C=0)$	0.05975698889837014	$P(X9,C=1)$	0.02569035181808912
$P(X10,C=0)$	0.03032820936293249	$P(X10,C=1)$	0.32077518089467844

Now, comparing the values(marked with blue), we let the highest in each row win ,follows that:

predicted class	actual class	
0	0	TN
1	0	FP
0	0	TN
1	0	FP
1	1	TP
1	1	TP
1	1	TP
1	1	TP
0	1	FN
1	1	TP

3)

F1 score = 0.7692307692307692
The F1 score is given by the following equation:

$$F1\ score = \frac{2*Recall*Precision}{Recall + Precision} \quad Precision = \frac{TP}{TP+FP} \quad Recall = \frac{TP}{TP+FN}$$

Recall $= \frac{5}{6} =$ 0.833333333

precision $= \frac{5}{7} = 0.7142857$

The result was obtained by applying the formula to the last diagram in 2)

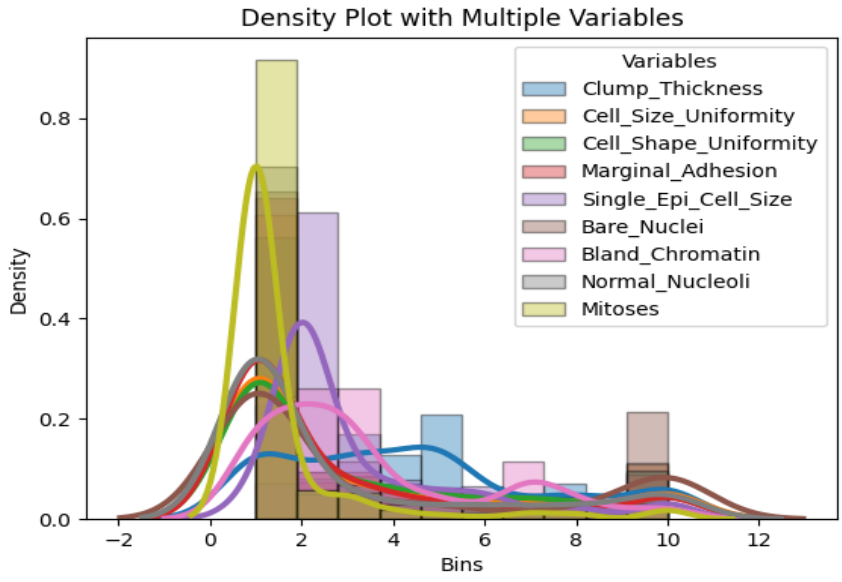
4)

Homework I – Group 72

P(class = 1 | X)		actual class	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
	X1	0	1										0
	X2	0	1										0
	X3	0	1										0
	X4	0	1										0
	X5	1	1										0
	X6	1	1										0
	X7	1	1										0
	X8	1	1										0
	X9	1	1										0
	X10	1	1										0
		accuracy											0

II. Programming and critical analysis

5)



6)

By applying the 10-fold cross validation to the given dataset. We obtained the following results for the accuracy of the data when applied to different values of k. For k=3 the accuracy is 0.9678815004262576, for k=5 the accuracy is 0.9708013640238706 and for k=7 the accuracy is 0.9722506393861892.

Despite, the results showing that the most accurate k is for k=7, nevertheless, being more likely to be the less to overfit k, there is a chance that, in reality, this k has a bigger risk than the other k's due to fitting more points, in order to get a better accuracy.

Moreover, the problem is that the fitting of the data doesn't distinguish noise points (points that are more distant than others) from regular points (points that are relatively close to each other), so there may be more noise points fitted just to get a better accuracy, with this also being noticed when k=5 having a lower accuracy than k=7 but still higher than k=3.

To conclude, its plausible to assume that the k with less risk of overfitting is k=3, due to having less noise points fitted giving it a lower accuracy overall but a better estimate of the data, since it doesn't fit as many points as the other k's with its overall risk of overfitting being lower than the other k's

7)

To verify the hypothesis "*k*NN is statistically superior to Naïve Bayes (multinomial assumption)". Let us take a t-test. It follows that H0 (the null hypothesis): KNN is statistically superior to Naive Bayes (multinomial assumption), H1 (alternative): KNN isn't statistically superior to Naive Bayes (multinomial assumption). We obtain the following values for the t-score and p-value respectively, 0.4876828319813145, 0.6374340829802527. From this data, we have enough evidence to conclude that since the t-score is below 50%, we accept the null hypothesis with 60% of confidence. Moreover, we can infer with a confidence of 60 % that the KNN is statistically superior to Naive Bayes (multinomial assumption).

8)

Two of the reasons that underlie the differences in performance between *k*NN and Naïve Bayes.

Firstly, *k*NN unlike Naïve Bayes doesn't require to train a model from the data and separate it in 10 folds (in this case) compare it with the various folds that are part of the test data.

Secondly, in this case, since our variables are conditionally distributed in the Naïve Bayes, we can suffer from the zero-probability problem, which happens when the conditional probability of a specific variable is equal to zero, failing to make good predictions because of it, this can explain the accuracy of the naïve bayes model being lower than the *k*NN due to not making good predictions.

III. APPENDIX

5.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
data = pd.read_csv('result-breast.csv')

vars =
['Clump_Thickness','Cell_Size_Uniformity','Cell_Shape_Uniformity','Marginal_Adhesion','Sing
le_Epi_Cell_Size','Bare_Nuclei','Bland_Chromatin','Normal_Nucleoli','Mitoses']
for var in vars:
    sns.distplot(data[var], hist=False, kde=True, bins=10,kde_kws={'linewidth': 3},label =
var)
plt.xlabel('Bins')
plt.legend(prop={'size': 10}, title = 'Variables')
plt.title('Density Plot with Multiple Variables')
plt.show()
```

6.

```
import numpy
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
f = open("data1.txt", "r")
lst = f.readlines()
resposta = []
for i in range(0, len(lst), 1):
    lst[i] = list(eval(lst[i]))
    resposta += [lst[i][-1]]
    lst[i] = lst[i][:-1]

LX = numpy.array(lst)
Ly = numpy.array(resposta)
X_train, X_test, y_train, y_test = train_test_split(LX,Ly, random_state=72)
knn = KNeighborsClassifier(n_neighbors=7)
scores = cross_val_score(knn, LX, Ly, cv=10, scoring='accuracy')
print(scores.mean())
```

7.

```
import numpy
import scipy
from scipy import stats
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from random import seed
from random import randrange
from math import sqrt
from math import exp
from math import pi

def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for _ in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split
```

```
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores =()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores

def separate_by_class(dataset):
    separated = dict()
    for i in range(len(dataset)):
        vector = dataset[i]
        class_value = vector[-1]
        if (class_value not in separated):
            separated[class_value] = list()
        separated[class_value].append(vector)
    return separated

def mean(numbers):
    return sum(numbers) / float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x - avg) ** 2 for x in numbers]) / float(len(numbers) - 1)
    return sqrt(variance)

def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del (summaries[-1])
    return summaries

def summarize_by_class(dataset):
    separated = separate_by_class(dataset)
    summaries = dict()
    for class_value, rows in separated.items():
        summaries[class_value] = summarize_dataset(rows)
    return summaries

def calculate_probability(x, mean, stdev):
    exponent = exp(-((x - mean) ** 2 / (2 * stdev ** 2)))
    return (1 / (sqrt(2 * pi) * stdev)) * exponent

def calculate_class_probabilities(summaries, row):
    total_rows = sum([summaries[label][0][2] for label in summaries])
    probabilities = dict()
    for class_value, class_summaries in summaries.items():
        probabilities[class_value] = summaries[class_value][0][2] / float(total_rows)
        for i in range(len(class_summaries)):
            mean, stdev, _ = class_summaries[i]
```

```
            probabilities[class_value] *= calculate_probability(row[i], mean, stdev)
    return probabilities

def predict(summaries, row):
    probabilities = calculate_class_probabilities(summaries, row)
    best_label, best_prob = None, -1
    for class_value, probability in probabilities.items():
        if best_label is None or probability > best_prob:
            best_prob = probability
            best_label = class_value
    return best_label

def naive_bayes(train, test):
    summarize = summarize_by_class(train)
    predictions = list()
    for row in test:
        output = predict(summarize, row)
        predictions.append(output)
    return (predictions)

def calcula_score1():
    f = open("data1.txt", "r")
    lst = f.readlines()
    resposta = []
    for i in range(0, len(lst), 1):
        lst[i] = list(eval(lst[i]))
        resposta += [lst[i][-1]]
        lst[i] = lst[i][:-1]
    LX = numpy.array(lst)
    Ly = numpy.array(resposta)
    X_train, X_test, y_train, y_test = train_test_split(LX,Ly, random_state=72)
    knn = KNeighborsClassifier(n_neighbors=3)
    scores = cross_val_score(knn, LX, Ly, cv=10, scoring='accuracy')
    f.close()
    return scores

def calcula_scores2():
    f = open("data1.txt", "r")
    dataset = f.readlines()
    for i in range(0, len(dataset), 1):
        dataset[i] = list(eval(dataset[i]))
    seed(72)
    n_folds = 10
    scores2 = evaluate_algorithm(dataset, naive_bayes, n_folds)
    for i in range(0,len(scores2),1):
        scores2[i] = scores2[i]/100
    f.close()
    return scores2

scores = calcula_score1()
scores2= calcula_scores2()
print(scores2)
scipy.stats.ttest_rel(scores,scores2, axis=0, nan_policy='propagate', alternative='two-
sided')
```